2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-6512-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/EuroSP57164.2023.00015

A Generic Obfuscation Framework for Preventing ML-Attacks on Strong-PUFs through Exploitation of DRAM-PUFs

Owen Millwood[1], Meltem Kurt Pehlivanoğlu[2], Aryan Mohammadi Pasikhani[1],
Jack Miskelly[3], Prosanta Gope[1], Elif Bilge Kavun[4]
[1]Department of Computer Science, The University of Sheffield, Sheffield, UK
{ojwmillwood1, a.mohammadipasikhani, p.gope}@sheffield.ac.uk
[2]Department of Computer Engineering, Kocaeli University, Kocaeli, Turkey
meltem.kurt@kocaeli.edu.tr
[3]Centre for Secure Information Technologies, Queen's University Belfast, Belfast, United Kingdom
j.miskelly@qub.ac.uk
[4]Secure Intelligent Systems Research Group, University of Passau, Passau, Germany
elif.kavun@uni-passau.de

***Abstract*—Considering the limited power and computational resources available, designing sufficiently secure systems for low-power devices is a difficult problem to tackle. With the ubiquitous adoption of the Internet of Things (IoT) not appearing to be slowing any time soon, resource-constrained security is more important than ever. Physical Unclonable Functions (PUFs) have gained momentum in recent years for their potential to enable strong security through the generation of unique identifiers based on entropy derived from unique manufacturing variations. Strong-PUFs, which are desirable for authentication protocols, have often been shown to be insecure to Machine Learning Modelling Attacks (ML-MA). Recently, some schemes have been proposed to enhance security against ML-MA through post-processing of the PUF; however, often, security is not sufficiently upheld, the scheme requires too large an additional overhead or key data must be insecurely stored in Non-Volatile Memory. In this work, we propose a generic framework for securing Strong-PUFs against ML-MA through obfuscation of challenge and response data by exploiting a DRAM-PUF to supplement a One-Way Function (OWF) which can be implemented using the available resources on an FPGA platform. Our proposed scheme enables reconfigurability, strong security and one-wayness. We conduct ML-MA using various classifiers to thoroughly evaluate the performance of our scheme across multiple 16-bit and 32-bit Arbiter-PUF (APUF) variants, showing our scheme reduces model accuracy to around 50% for each PUF (random guessing) and evaluate the properties of the final responses, demonstrating that ideal uniformity and uniqueness are maintained. Even though we demonstrate our proposal through a DRAM-PUF, our scheme can be extended to work with memory-based PUFs in general.**

***Index Terms*—Physical Unclonable Functions, Strong PUF, DRAM-PUF, Machine-Learning Modelling Attack, PUF Obfuscation**

1. Introduction

Physical Unclonable Functions (PUFs) are security primitives which are used to derive secret information from intrinsic hardware properties without storing any explicit data [27]. Sub-atomic variation in circuit structure caused by manufacturing process variation results in nominally identical chips exhibiting unique characteristics, which can be exploited to provide an identity strongly tied to the physical system. This is analogous to biometric authentication for humans, where small physical (genetic) variation results in unique, measurable characteristics at the macro level. PUFs can be used for memory-less key storage in a cryptographic system, direct authentication in a challenge-response protocol, or as a lightweight unique device ID.

One of the most significant concepts in PUF research is the division between so-called "Strong" and "Weak" PUFs. This is based on the pairs of input challenges to output responses (Challenge-Response Pairs, or CRPs). Strong PUFs have a large set of CRPs, which grow rapidly with increasing circuit size. On the one hand, this is advantageous, allowing for large amounts of secret information to be generated from a very small hardware footprint. The corollary is that due to the relatively small number of significant physical variables underlying the PUF behaviour, they can be vulnerable to modeling, as the discernment of these variables allows for the prediction of the response to any arbitrary challenge within the total set, i.e., a mathematical "clone" of the target PUF. As each physical variable contributes to many responses, derivation of the variables is often possible with only a small subset of captured CRPs. Conversely, Weak PUFs have a small set of responses (often just one), which increase linearly with circuit size. In most Weak PUF designs, each bit of the response is determined by a distinct physical variable such that knowledge of part of the response provides no information about the variables underlying the remainder of the response. This leads to a reverse of Strong PUF properties – no modelling vulnerability but a far less desirable ratio of circuitry to secret information. Ironically,

DOI 10.1109/EuroSP57164.2023.00015

"Strong" PUFs are highly flexible in their uses within a protocol but weak to trivial attacks. In contrast, "Weak" PUFs are more limited in their use cases but are much more difficult to compromise.

The use of Strong PUFs is most commonly associated with authentication protocols [15], [36], [41], where the large number of CRPs allows for individual authentication tokens to used once then discarded without running out of possible tokens quickly, such that a replay attack cannot be mounted by an attacker trying to impersonate a device. Such protocols can, in theory, be far more lightweight than most cryptography but are entirely reliant on the security of the underlying PUF. Weak PUFs on the other hand are most commonly associated with key storage, where key material for encryption can be generated from a PUF circuit without the need for explicit key storage in a Non-Volatile Memory (NVM) [27]. Many Weak PUF designs use structures similar to those found in memory, or derive a PUF response from existing volatile memory circuits (e.g., SRAM, DRAM). This makes them well suited to the generation of large blocks of secret information at minimal resource cost.

The first and arguably most well documented type of a Strong PUFs are Arbiter-PUFs (APUFs), which exploit intrinsic delay variation in two nominally identical electrical pathways, which are influenced by chained multiplexers and are formalised as an additive linear delay model (Figure 1) [12]. As with all Strong PUFs proposed to date, APUFs are vulnerable to Machine Learning (ML) based Modelling Attacks (ML-MA) [3], [34], [35]. This is because in a Strong PUF the large set of CRPs arises from a relatively small number of actual physical variables. i.e. the PUF function is rooted in complex physical factors, but is mathematically simple. Even for a large PUF, if some sufficient subset of CRPs is known it takes relatively little computation to derive a function which produces the same outputs as the actual PUF for any given challenge. A range of variants have been proposed in an attempt to reduce the linearity and correlation between the initial challenge data and response data such as the XOR-Arbiter-PUF (XOR-APUF) [36], Interpose PUF (iPUF) [33], Feed-forward Arbiter PUF (FF-APUF) [14] and Lightweight-Secure PUF (LS-PUF) [29]. While each APUF variant showed an initially enhanced resilience, each was eventually shown to be vulnerable to ML attacks which determined the more complex correlative properties between challenges and responses [34], [37], [39]. How, or if, Strong PUF designs like the Arbiter-PUF can be reliably secured against ML-MA without introducing excessive complexity remains an open question.

Ideally, there are a number of desirable properties (DP) for a PUF obfuscation scheme to exhibit:

- **DP1: Minimal Area Requirement**:- PUF obfuscation requires additional logic and/or operations on top of the PUF circuit itself. Thus, to keep the obfuscated PUF lightweight and applicable to constrained systems the obfuscation mechanism must use the lowest amount of system resources possible. Additional circuitry should be minimised for power and area, computation should be kept to a minimum, and existing system resources repurposed as much as possible.

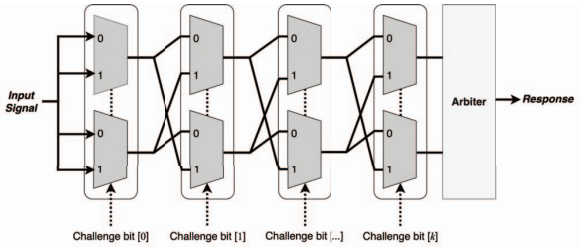

Figure 1: Arbiter-PUF

- **DP2: ML-MA Secure**:- The scheme must provide sufficient non-linearity between PUF inputs and outputs (CRPs) such that an attacker with knowledge of part of the CRP set cannot mount a modelling attack using machine learning that results in a prediction rate significantly above 50% for CRPs not in the known set.
- **DP3: No NVM Requirement**:- A common requirement in authentication systems and in previously proposed PUF obfuscation schemes (such as [42]) is an NVM to store secrets or helper data on the device. Having a secret or data which reveals compromising properties of the system constantly present on the device during operation is not ideal and creates an obvious point of attack for adversaries. Further, in a PUF context it weakens a key property of the PUF, that the PUF secrets exist only when being used. A PUF obfuscation scheme, therefore, should try and avoid an NVM requirement and in general avoid the need for fixed additional secrets or helper data which contains information about the PUF properties. If supplementary data must be used, it should not give the adversary any information about the PUF properties or behaviour.
- **DP4: Reconfigurable**:- Ideally, in addition to passively preventing ML-MA by raising the difficulty of modelling an obfuscation system should contain a mechanism for reconfiguration. i.e. changing the system behaviour such that using the same challenge twice between different configurations exhibits different final outputs. This provides an element of active countermeasure. If an attack is suspected the PUF can be reconfigured, and a model trained to predict the previous configuration will now need to be retrained.
- **DP5: Generic**:- The scheme should be designed separately from any given PUF design, such that designers can use any arbitrary PUF in the scheme and produce the same enhanced security. This kind of modularity ensures that the obfuscation scheme can be tested on future-developed PUFs and be implemented to secure any strong PUF which a given manufacturer desires. This enables use of previously designed PUFs with specific desirable properties, such as high reliability in the case of shorter length Arbiter PUFs, but which are vulnerable to ML-MA without additional obfuscation logic [17]. A truly general scheme gives the designer the freedom to tailor the properties to the needs of the application.

1.1. Related Work

Several schemes have been proposed that aim to fortify the security properties of Strong PUFs. The fundamental core of all ML-MA is the fact that the relationship between a PUF challenge and the matching response is the PUF function itself. Given a training set of known CRPs it is possible to infer a model which predicts the behaviour of the PUF function for all challenges, including those not in the training set. Therefore, anti-modelling countermeasures aim to obfuscate this relationship and, by extension, the PUF function. Ideally this would render ML-MA impossible, but in practice, many methods simply aim to raise the modelling complexity enough to make attacks impractical in the field. Obfuscation methods can mainly be categorised in two ways with subtle differences: structural non-linearisation and CRP obfuscation. Structural non-linearisation involves designing PUFs where the actual PUF function and challenge-response relationship are less linear in structure, thus increasing the modelling complexity [14], [25]. CRP obfuscation aims to obscure the challenge and responses via masking, hash functions, etc., so CRPs cannot be used to infer the PUF function unless the adversary can also reverse the obfuscation function [11], [13], [36].

In 2004, Gassend et al. introduced the FF-APUF, whereby the results of early PUF stages are fed-forward to several challenge inputs further along, reducing the linearity of the PUF [14]. While in practice, this PUF showed strong resistance to Linear Regression (LR) based ML-MA, Alkatheiri et al. demonstrated a successful attack using a Multi-Layer Perceptron (MLP) to model the FF-APUF [1]. In 2007, Suh et al. proposed the first PUF to include CRP obfuscation in the XOR-APUF, where the output of a number of unique APUFs is XOR'ed together to obscure the mapping of each individual APUFs responses against the same challenge [36]. This additional logic (and added resistance to ML-MA) came at the expense of reduced PUF reliability and increased hardware overhead. Ma et al., Miskelly et al., and Cui et al. proposed the use of lightweight single-cell Weak PUFs as an alternative method of CRP obfuscation [28] [31] [6], where each CRP bit is XOR'ed with the response of a single bit Weak PUF. The hardware overhead of this approach is minimal, but in practice only reduces the prediction rate of advanced attacks with large CRP training sets to around 80%. This gives the adversary a significantly increased chance of achieving a collision, even if it prevents fully reliable prediction.

Gassend et al. proposed a Controlled PUF, whereby hash functions are utilised to restrict access to CRPs [13]. The combination of multiple hash functions and error-correction code (ECC) incurs a substantial hardware requirement, making it difficult to justify the PUF over traditional cryptographic methods. Ye et al proposed an RPUF in [40], whereby each challenge is randomized before being input to the PUF itself. This scheme however showed insufficient resilience against ML-MA with most attacks providing above 70% prediction accuracy. Subsequent attacks increased this to above 90% [8]. In [11], Gao et al presented PUF-FSM, where ECC is replaced with a finite-state machine, however, the hash logic still required by the scheme ensured the hardware overhead was not sufficiently low and the selection for reliable responses in the CRP set provides enough information leakage to mount a successful attack as in [8]. Dubrova et al proposed the CRC-PUF, utilising a circuit based on Cyclic Redundancy Checking to perform a transformation of the PUF input in order to increase the difficulty of ML-MA [9]. While compact and generically applicable, the efficacy of this solution is not clear. Only one ML-MA method was tested experimentally, LR, and even then a prediction rate of 75% per-bit was achieved. It is also not a complete scheme as it relies on the generation of a polynomial to configure an LFSR each time the PUF is used, but no specific method for generation or handling at the protocol level is proposed and this is not considered in the resource usage.

Recently, Zhang et al proposed an obfuscation scheme for Strong PUFs which utilises pre-stored stable PUF responses to use for obfuscation after enrollment [42]. A TRNG is used to select randomly from a set of keys to obscure CRPs using XOR operations, and when a number of CRPs is able to be collected by an adversary, the set is updated. While this scheme enabled strong resilience against ML-MA and lower hardware overhead than similar schemes, it relies on storing key data in NVM, enabling adversaries with physical access to gain access to the secret data, which would ultimately compromise the security of the PUF. Zhang et al also proposed a scheme which uses a structure that can operate as three different kinds of PUF (arbiter, ring oscillator and bi-stable ring), called CT-PUF [43]. In order to reduce the linearity between challenges and responses, the CT-PUF acts as only one of these PUFs to any given challenge, but the challenge itself is also obscured by being fed through an arbiter PUF. The CT-PUF structure is more area efficient than other proposals as the single circuit provides all three PUF functionalities. It achieves very good, but not perfect, ML-MA resistance with prediction rates just over 60%. However, it is not a generalised scheme that can apply to any strong PUF. Additionally, while it is impressively compact, it will be shown in this work that even greater area reductions can be achieved through re-use of existing components.

1.2. Comparison

Table 1 shows a comparison of the relevant works presented against the identified desirable properties. The key distinguishing factors are DP1 and DP5. The proposed scheme is generic and enables the use of any type of strong PUF in a modular way, while other schemes are limited in this regard. The proposed scheme achieves *DP5* by completely separating the chosen PUF from the rest of the obfuscation logic, such that any strong PUF output is fully obfuscated with the lightweight proposed one-way function (described in detail in Section 2.3). While some other schemes do achieve this generic quality they end up using more resources than the design proposed in this work. The proposed scheme achieves this reduction (*DP1*) by utilising existing resources on device in the form of memory PUFs, which enable entropy generation in runtime without additional logic of use of permanently storing data in NVMs.

TABLE 1: Comparison of the Proposed Scheme Against Similar Schemes

Scheme	*DP1*	*DP2*	*DP3*	*DP4*	*DP5*
Controlled PUF [13]	X	∼	X	✓∗	✓
PUF-FSM [11]	X	X	✓	✓∗	✓
Set-based [42]	X	✓	X	✓	X
CRC-PUF [9]	∼	∼	✓	✓	✓
CT-PUF [43]	X	✓	✓	✓	X
Proposed Scheme	✓	✓	✓	✓	✓

✓: Yes; X: No; ∼: Inconclusive/Not Explicitly Evaluated
∗ If hash function used is reconfigurable
$DP1$: **Minimal Area** $DP2$: **ML Secure** $DP3$: **No NVM Required** $DP4$: **Reconfigurable** $DP5$ **Generic Framework**

1.3. Motivation

The major issue with Strong PUF obfuscation schemes is not in devising the methods of obfuscation - any number of known hard problems can be applied to increase the modelling complexity to an infeasible degree. The real problem is: how do we do this in a way that retains the lightweight nature of the PUF concept? If the obfuscated system requires many aspects of traditional cryptography, most of the benefits of a Strong PUF over purely cryptographic functions are lost. There is still the fact that it is rooted in hardware, but that benefit is not unique to Strong PUFs by any means. The key goal, then is to find the method(s) which add the most complexity for the lowest cost. For most schemes there are three inescapable overheads in addition to the PUF itself:

(a) Some kind of non-reversible function or mask to decorrelate challenges from responses.
(b) A fixed matrix or secret which can be used in that function and an NVM to keep it in. This can sometimes be avoided depending on the nature of (a), but for very lightweight functions this is generally a requirement.
(c) Strict error correction, necessary due to the error amplifying nature of most applicable functions and inherent PUF noise.

As error correction is always going to be necessary in PUF based systems, the two targets for overhead reduction are the masking function (a) and the fixed secret/NVM (b), but schemes which avoid these have less than ideal results. Schemes which rely on more complex PUF structures alone, avoiding (a) and (b), have thus far largely proven vulnerable to more sophisticated ML-MA. Schemes which rely on very lightweight masking instead of a hash or one-way function, avoiding (b), only partially impede attacks. Steady progress has been made on reducing the footprint of masking functions but overall footprint remains higher than is desirable especially in systems using NVMs (which are relatively expensive, bit-for-bit). NVMs also present something of an obvious target for hardware level attacks as they typically present a single point of failure for the obfuscation system.

Something less explored in this context is the fact that PUFs are themselves a type of 'memoryless' key storage mechanism, often used in place of NVM. For most PUFs this results in no net gain in cost, as the PUF would need as much hardware as an NVM while being less stable. However, if we consider a software PUF based on a volatile memory, this becomes a much more interesting proposition. It is safe to assume most systems will have at least one volatile memory. If this memory may also be exploited as a PUF, a concept which has been demonstrated on both SRAM [21] and DRAM [22] [23] [32], then it can be used as a matrix generator to remove the need for NVM entirely. If such a memory PUF can be operated during system runtime as is the case for [23] and [32], it also provides the advantage of the fixed secret only existing when needed, rather than being constantly present on the device. Further, the size of a system's main memory will naturally be much larger than what could be justified for NVM in a single sub-system, allowing for a larger fixed secret than would otherwise be feasible. In addition to the issue of resource use, there is a practical consideration around the diversity of PUF designs proposed to date. An obfuscation scheme based around a particular Strong PUF is helpful but not as useful as a generic scheme into which any existing Strong PUF IP can be inserted. However, a generic scheme requires more rigorous testing because the modelling complexity of the PUF function is not fixed. Scheme resistance to one ML attack for one PUF type does not necessarily extrapolate to equal resistance to any given ML attack for any given PUF type. This is less of a concern for rudimentary attacks but more crucial when considering adversarial learning, deep neural nets, evolutionary strategies, and similar advanced ML techniques, which are well suited to finding fits for highly complex functions. Ideally, a scheme should be both generic and tested experimentally against a broad range of ML techniques for various complex PUFs.

1.4. Contributions

To provide a solution to the limitations identified in the current literature, we propose a generic DRAM-PUF-based obfuscation scheme. The main contributions of this paper are as follows:

- A generic PUF-based obfuscation scheme for any Strong PUF, experimentally verified using multiple 16-stage and 32-stage Arbiter-based PUF variants simulated in software. To the best of our knowledge we are the *first* to consider a modular scheme with regard to Strong PUF where an adversary is considered to have full knowledge of the utilised underlying scheme.

- A novel modified One-Way Function (OWF) to enable strong resilience to ML-MA and enhanced one-wayness to our scheme, with no impact on ideal Uniqueness and Hamming Distance properties. In this regard, we utilise DRAM-PUFs for OWF configuration, having the effect of improved security with reduced hardware footprint.

- An evaluation of our scheme against well-known Machine Learning Modelling Attacks against strong PUFs, including a novel set of supervised and unsupervised classifiers tailored for our scheme, showing a strong capability to resist ML-MA.

- Synthesis of our OWF on a Zynq-7000 FPGA and provide a comparison of our proposed scheme for hardware overhead and power consumption against

comparable schemes.

- We provide all source code and data used for our experiments, open access for the research community (Section 6).

1.5. Paper Organisation

The remainder of the paper is organised as follows. Section 2 introduces preliminary information regarding Memory-PUFs, One-Way Functions and notation/technical concepts used throughout the paper. Section 3 provides a detailed description of the operation of the proposed scheme. Section 4 explains the experimental methodology undertaken to evaluate the proposed scheme. Section 5 then provides a breakdown and analysis of our experimental results including a comparison against similar schemes and finally in Section 6, we provide a conclusion of this work.

2. Preliminaries

In this section, we outline some preliminary concepts pertinent to the proposed scheme and evaluation methods.

2.1. Memory-PUFs

PUFs are not limited to being comprised of discrete PUF-specific circuitry. Software PUFs have been proposed that utilise software controls to measure entropy present in hardware which already exists on a device. Memory such as Static Random Access Memory (SRAM) and Dynamic Random Access Memory (DRAM) are key targets for PUFs of this type and have produced some excellent results [21], [22]. In these designs, each memory cell produces a single response bit, typically producing one very large PUF response equal to the storage size of the memory module itself. As such, they fall firmly into the category of Weak PUFs. This work makes use of a type of DRAM-based memory PUF called a DRAM Latency PUF. A commodity DRAM will begin to experience latency-induced errors if insufficient time is given for certain internal operations. This is due to the inherent latencies of the physical processes involved, e.g., capacitor charge and discharge, voltage adjustments, amplification, etc. Through a combination of clock and memory controller manipulation, it is possible to place a DRAM module in a state where the deciding factor in whether any given operation on a specific memory address succeeds or fails is the process variation in the cells being operated on. PUFs of this type have been shown in practice to produce secrets with strong security properties [23], [32]. In addition, they have several desirable properties: they can be implemented on existing commodity systems, can generate keys as needed in a just-in-time manner, have a maximum key size measured in the 100s of MB to GB, and operate at high speed - under ideal circumstances with a shorter delay than reading a fixed key from the same memory [32].

2.2. Machine Learning Modelling Attacks on PUFs

ML-MA have been a primary security issue for Strong PUFs almost since original conception. In such attacks, the ML algorithm collects and analyses the Challenge/Response Pairs (CRPs) from a given PUF and uses them to create a model of the target component which exhibits identical behaviour. Earlier PUF types have been shown to be easily broken by [1], [3], [34]. Given a model which successfully plots the linear (or in some cases non-linear) feature space of the PUF function, an attacker is able to eavesdrop on novel, unspent challenges, predict the corresponding responses and mount a man-in-the-middle attack in order to impersonate the target device, breaking the security of the PUF (and thus often the entire scheme).

2.3. Lightweight One-Way Function

In this work, we modify the compression function given in [10] (it is the improved variant of the function proposed in [2]) to design our One-Way Function (OWF). Here, we use our own truly random generated H matrices by using DRAM-PUF data instead of a random quasi-cyclic matrix in order to ensure optimal security. Contrary to what has been reported in [10], our construction is suitable for memory-constrained environments. In the original construction of the compression function, as described in [2], denoted F, takes s bits of input data, and uses a random $r \times n$ binary matrix to obtain r bits of output data. F consists of XOR operations and simply computes the syndrome of the split parts (i.e., s bits of data are split into blocks of w, and then w columns of H are XORed). The inverting the F is a Syndrome Decoding problem that is NP-complete [4]. It proves that when you choose the appropriate parameters, this compression function can be used as an OWF.

In [10], the authors improved the previous original construction [2] by using a quasi-cyclic matrix H instead of the generic random matrix to increase the overall efficiency. But still, this improved compression function does have its limitations, for example, the size of H is still a limiting factor for the efficiency of the compression function. Furthermore, the quasi-cyclic codes cannot guarantee complete randomness [10]. Therefore in this work we focus on how we can generate a truly random small binary matrix H of 8,192 bytes (note that here the parameters $r = 64$, and $n = 1024$) with appropriate w parameters, without any security loss for designing our OWF.

Entropy Enhancement and Resourcefulness for H-Matrices. A basic approach to the the H matrix requirement in hardware would be to generate a cryptographically significant matrix (through an arbitrary PRNG/TRNG) during device enrollment and permanently store it in a form of NVM to be accessed when required. This, as mentioned previously, creates a significant weakness for a PUF device, where it is assumed an adversary has physical access and could read H in this scenario which would entirely undermine the security of the PUF scheme, not to mention the significant hardware overhead incurred for each bit of required NVM. For

these reasons, we focus on the benefits of exploiting Memory-PUFs which are already suitable to utilise available device resources to generate large amounts of high entropy data (discussed further in Section 4.4).

By modifying the idea given in [10], the steps for the proposed OWF can be written as follows:

1) Select block number w,
2) Split m bits of input data into w blocks, by the way, if padding is required pad it with the padding method,
3) Convert each of the blocks from binary to integer $(x_1, x_2 \cdots, x_w)$,
4) Pick the corresponding column in the H matrix at position x,
5) Return $C_{Final} = H_{x_1} \oplus H_{x_2} \oplus \cdots \oplus H_{x_w}$

In our OWF, according to the chosen value of w, m needs to be padded. We check whether m is divisible by w, if it has no remainder the padding is not necessary, otherwise it needs. For the padding, it is inspired by PKCS#7 [24], but we made some changes: the number of required padding bits is to be calculated with the following formula:

$$N_{padding} = ([(q_1 + 1) \times w] - m) \tag{1}$$

where q_1 is the quotient of m and w. Then, the result is converted to a 4-bit binary number, and we repeat it y times. Let q_2 be the quotient of $N_{padding}$ and 4, if $N_{padding}$ is divisible by 4 without remainder, y is equal to q_2, otherwise, will be equal to (q_2+1). As a final step, we drop the last $(4 * y - N_{padding})$ bits to get the padded input data. For example, let w be 7, in here we need $6 (= [(9 + 1) \times 7] - 64)$ bits padding, the 4-bit binary representation of 6 is "0110", then we repeat "0110" $2(= 1 + 1)$ times as "01100110". We only need 6-bit padding so we remove the last $2(= (4\times2)-6)$ bits, finally, the pad will be "011001". When evaluating the hardware cost of the OWF function, in theory, splitting m has no cost. Here, for the last step costs only $[(w - 1) \times m]$ binary XORs.

According to Algorithm 1, the concatenation of bit strings M and $pad(M, w)$ is denoted by $M \| pad(M, w)$ where $M \in \{0, 1\}^{64}$ is an 64-bit string and $pad()$ is a padding function. When x is an integer, we write $\langle m_i \rangle_b$ to represent its corresponding b-bit binary representation.

Algorithm 1: One-Way Function (OWF)

Input: $M : R_{Origin}$
Data: w: *Block number*
Output: C_{Final}

$M' \leftarrow M \| pad(M, w)$
$m_1 \| \cdots \| m_w \leftarrow$ ***parse*** (M')
/* where $|m_i| = b$-bit for all $1 \leq i \leq w$ and *parse*() is a function that splits M' into w blocks */
$x_i \leftarrow \langle m_i \rangle_b$
/* where x_i is the integer representation of b-bit bit stream m_i */
for $i \leftarrow 1$ *to* w **do**
  $H_{x_i} \leftarrow$ ***pick***(H, x_i)
  /* where *pick*() is a function that picks the corresponding column in the H matrix at position x_i */
  $C_{Final} \leftarrow (C_{Final} \oplus H_{x_i})$
return C_{Final}

2.4. Notation

Hamming Distance. It is essential that the OWF does not tend towards a bias for outputting more than one bit over another. We evaluate this quality using the hamming distance (HD). The number of bit positions where the two bits differ gives the HD and the HD of m-bit X and Y is defined as:

$$HD(X, Y) = \sum_{i=0}^{m-1} X[i] \oplus Y[i] \tag{2}$$

Uniformity. Under the different challenges, the uniformity metric gives the distribution of "0"s and "1"s in PUF responses. For the n different m-bit PUF responses (represented by R), the uniformity is defined as:

$$Uniformity = (\frac{1}{n} \sum_{i=1}^{m} R_i) \times 100\% \tag{3}$$

2.5. Threat Model

This work is based on a threat model which assumes a *remote adversary* with the technical ability and resources to carry out advanced ML-MA. Specifically, the following assumptions are made:

1) The adversary is remote, or has limited physical access to the target device. While physical level attacks against PUFs are a factor of consideration they are not the primary motivator of this work.
2) The adversary has a goal of being able to predict the final output, R_{Final}, for any given challenge, C_{Origin}.
3) Any initial challenge, C_{Origin}, and corresponding R_{Final} used by a device are available to the adversary via eavesdropping. It is assumed the attacker can always acquire a moderately sized subset of the possible CRPs for any given device.
4) The internal feedback challenge and response, C_{Final} and R_{Origin}, are not available to the adversary and cannot be directly measured. The adversary may attempt to predict them as a step in the ML-MA process but has no way to verify this except through changes to the predicted R_{Final}.
5) The adversary has full knowledge of the scheme structure.
6) The adversary has full knowledge of the OWF structure and can execute a copy of it at will.
7) The adversary has full knowledge of which PUF designs have been used and the mechanisms they employ.
8) The adversary can send falsified challenges to the scheme as a whole but cannot bypass the scheme to query either the internal SPUF or Memory-PUF individually.

There is an argument to be made for the inclusion of side channel analysis, hardware tampering, and/or fault injection as would be possible for a sufficiently advanced

adversary with physical access. While not to be dismissed, experimental analysis of these attacks is beyond the scope of this work which is focused on the ML-MA threat. Some preliminary discussion of the implication of these other attacks for the proposed scheme is given in Section 5.2 with further exploration likely to form the basis of future work.

3. Proposed Scheme

For an adversary to collect a subset of possible CRPs and accurately model the PUF behaviour, there must exist a reasonably strong correlative relationship between the initial challenge and final response data. Based on this assumption, we aim to complicate the mapping between initial challenges and final responses collected by an adversary such that resilience against ML-MA is sufficiently enhanced. Additionally, we consider the requirement to maintain a small enough hardware footprint to enable a scheme that can be reasonably deployed on edge devices.

We, therefore, present a generic obfuscation scheme where the Strong PUF element is considered modular, such that various Strong PUF implementations may be used interchangeably to generate hardware-centric tokens for further processing. To limit the overall footprint of a PUF obfuscation scheme it is desirable to enhance entropy using existing resources where possible. We achieve this in two ways: Firstly, while Strong PUFs on their own are susceptible to ML-MA, there is still an amount of entropy which is generated from the manufacturing variation of the PUF for each CRP. We exploit this by introducing a self-feedback feature where initial Strong PUF responses are used to create a new obfuscated challenge for the same Strong PUF. This contributes a baseline increase in modeling complexity for a remote adversary, as they will not have access to the internal feedback challenge. It adds very little hardware and is a generic solution which can be applied to any strong PUF. Further, it provides some flexibility in that depending on the stability of the PUF and acceptable level of resource consumption the number of feedback rounds can be increased or decreased.

In itself this increase in complexity is not sufficient to prevent ML-MA, however. Therefore, we also propose to use a low-cost OWF (as described in Section 2.3) in the feedback loop, such that the relationship between the origin challenge and the internal challenge becomes non-reversible and extremely difficult to predict without knowledge of a matrix, H, and the internal PUF response, R_{Origin}. In addition, we propose to exploit a second PUF - a Memory-PUF - as a matrix generator to supply the OWF with i.i.d matrices.

Generating matrix data in this way has key benefits. The OWF requires H to be a large random matrix, which would typically require permanent storage on-device. This both increases the scheme footprint and provides an obvious target for hardware level attacks. By offloading the synthesis and storage of H data to a Memory-PUF, both limitations can be mitigated. In terms of cost, the Memory-PUF can generate very large matrices with the necessary properties from its own hardware. In the case of the exemplary DRAM-PUF used in this work a significant proportion of the memory tested was usable for matrix generation, giving a total matrix space in the order of 100s of MB per GB of total DRAM (see section 4.4 for further detail). An equivalently sized fixed matrix with NVM storage would incur a substantial hardware and processing cost.

In combination these measures provide a very significant increase to the modeling complexity while using minimal additional circuitry. The specifics of the scheme are given in the following section, with tests of attack resiliency, resource consumption, and security analysis provided in Sections 4 and 5.

3.1. Generic PUF Obfuscation

We present our proposed generic obfuscation scheme, the entire process of which is depicted in Figure 2 and is described with the following numbered steps.

Response format: A binary vector of length i.

Challenge format: The SPUF is assumed to generate 1 bit responses to x bit challenges, where x is typically the number of SPUF stages. The OWF requires a fixed 64-bit input, meaning $64x$ challenge bits are required for one execution of the OWF. This will produce a 64-bit output, which can be split into $64/x$ feedback challenges to produce $64/x$ bits of the final response. This must be repeated until the desired response size, i, is reached. Therefore the expected challenge C will consist of the full set of required SPUF challenges, C_{Origin}, concatenated with the Memory-PUF challenge C_{Mem}. C_{Origin} will be of length $(64x) * (i/(64/x))$ bits.

① First, the challenge data C is received by the device and split into the SPUF challenge C_{Origin} and Memory-PUF challenge C_{Mem}. The first $64x$ bits of C_{Origin} are taken as C^i_{Origin}.

② C^i_{Origin} is passed to the SPUF to generate the internal response, R^i_{Origin}.

③ Error correction is performed on R^i_{Origin} to remove noise. PUFs contain an inherent degree of random noise largely influenced by environmental condition variation. Therefore, error correction is performed on the PUF output to remove this noise and generate a stable output. The scheme does not specify a correction method as the optimal approach will vary depending on the particular properties of the chosen PUF (an example however is provided in Section 5.3 to demonstrate hardware overhead).

④ If $i <= 64$ (i.e., the scheme is in the internal feedback stage) R^i_{Origin} is appended to a secure 64-bit register where the full R_{Origin} is assembled.

⑤ The secure register waits for a full 64 bit R_{Origin}. This register must be carefully secured as whole or partial leakage of R_{Origin} invalidates the ML-MA countermeasures.

⑥ The completed R_{Origin} is passed to the OWF, which now requires the configuration matrix, H.

⑦ The Memory-PUF challenge C_{Mem} is passed to the Memory PUF controller, which prepares the memory segments indicated in C_{Mem} for PUF use.

⑧ The Memory-PUF generates a preliminary (noisy) response matrix.

⑨ Error correction is performed on the Memory-PUF output to produce the final de-noised matrix, H. Provided there is sufficient memory space available, this

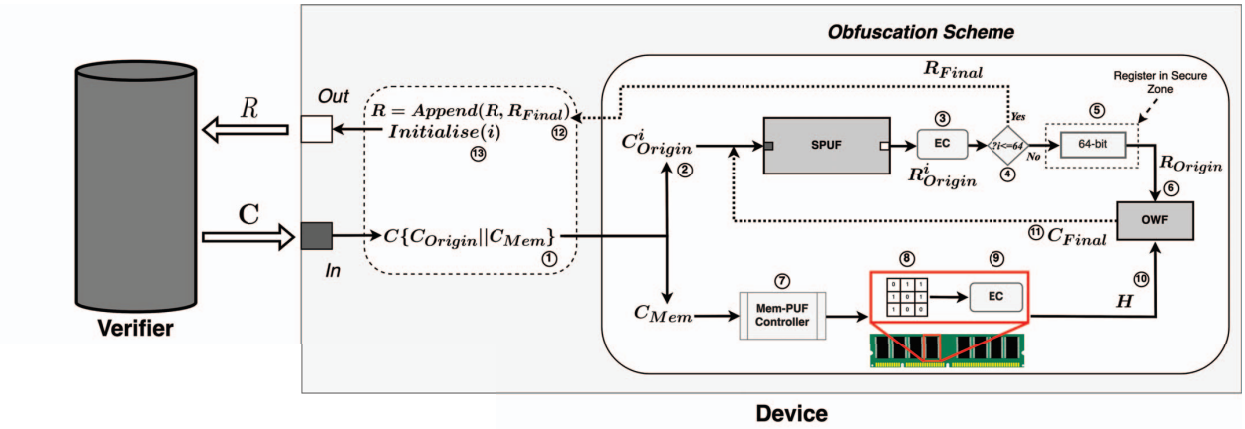

Figure 2: Generic PUF Obfuscation Scheme

is carried out also within the memory (to store a copy for majority voting over repeated measurements).

(10) H is used to configure the OWF, which has already been passed R_{Origin}. The 64 bit output of the OWF is the internal challenge, C_{Final}.

(11) C_{Final} is divided into $64/x$ vectors of length x, which are fed sequentially as challenges into the SPUF.

(12) The output bits are again error corrected to remove noise, then the R_{Final} bits are appended to the final response R.

(13) Finally, i is re-initialised to 0 and the process is repeated, using the next $64x$ bits of C_{Origin} as C^{i+1}_{Origin}. This continue until the required size of R is reached, after which R is returned to the verifier.

4. Experimental Methodology and Discussion

In this section, we first provide details of the setup/methodology used to validate the proposed obfuscation scheme, including details on the PUF datasets, quality of DRAM-PUF measurements and threat model.

4.1. Strong PUF Datasets

As with most hardware cryptographic primitives, adjusting PUF parameters can enable a designer to tailor the PUF design around the properties most desired for a given security scheme. With Arbiter-based PUFs, one such parameter which can be varied is length of the PUF (referred to as 'stages') which is adjusted through variation of the number of multiplexers used in the delay chain (see Figure 1). Increasing the size of the PUF has an exponential effect on increasing or decreasing the available CRP space, in theory enhancing the security of the PUF at the cost of decreased reliability. As shown in [34], while longer APUFs require more resources to model with ML-MA, APUFs of even 128-bits in length can be broken fairly trivially with sufficient CRP training data. While some may argue these render such PUFs redundant, we propose it as a reason to utilise the PUF for its strength in ability to generate many unique CRPs on-the-fly without any kind of additional processing, while security against ML is offloaded to other mechanisms in the overall scheme. For this reason, we propose the use of smaller APUFs, and thus verify our proposed scheme on 16-stage and 32-stage APUF variants, where the 16-stage APUFs supports 2^{16} unique CRPs and the 32-stage APUFs support 2^{32} CRPs. This design choice intrinsically reduces hardware overhead simply due to the smaller PUF size, but also inevitably reduces further error correction overheads as PUF reliability is not worsened with a larger PUF. For a 16-stage PUF, while it may seem only to support a relatively low number of possible unique CRPs (65,536), our scheme resolves this issue through intrinsic support for reconfigurability. As the challenge/response behaviour of the entire scheme is dependent on both the Strong-PUF challenge and the unique matrix, H from the Memory-PUF, the number of possible Strong-PUF CRPs scales directly with the number of unique responses supported by the Memory-PUF. In this way, just 10 unique Memory-PUF responses brings the total supported CRPs of the proposed scheme to 655,360 CRPs, providing an enhancement for scalability. Additionally, this has the effect of improved resistance to ML-MA as when the scheme is reconfigured, an adversary must attempt to train a new ML model to capture the new CRP behaviour of the scheme (discussed further in Section 5.5).

We utilised the PyPuf Python framework to generate a set of Strong PUF datasets, through simulating the linear delay models of each PUF tested [38]. To cover a variety of APUF variants, we tested both 16-stage and 32-stage APUF, XOR-APUF and FF-APUF. These PUFs have well defined mathematical models which describe their behaviour and can therefore be accurately simulated in this context. These simulations may not precisely emulate the behaviour of specific hardware implementations of these PUFs, but they do accurately represent the general behaviour and, crucially, their modelling complexity. As this work aims to evaluate a counter-modelling scheme and the simulated PUFs are an accurate representation of the work needed to perform a modelling attack they are entirely suitable for use here.

4.2. Memory PUF Dataset

To generate the H-matrices for our OWF, we opted to use a Latency DRAM-PUF due to the size of available responses, ideal PUF properties, high measurement speed and the ubiquity of DRAM in commodity devices. It should be noted that this is only a test case using a particularly well suited PUF design. In theory, any weak PUF which can quickly generate large responses could perform the same function. Unlike the strong PUF designs used, DRAM based PUF behaviour has only been observed experimentally. There is no defined mathematical model by which the physical level behaviour can be simulated. It would be possible to simply assign each simulated cell a probability of failure, but this misses many nuances such as the influence of surrounding cell contents, line variation, sense amplifier variation, etc. Developing and verifying the accuracy of such a model is beyond the scope of this work, therefore we chose to source DRAM PUF data generated experimentally from hardware. The experimental setup for this used the same setup described in [30], targeting Commodity Off-The-Shelf (COTS) DIMM form factor desktop DRAM modules. In each experiment a data pattern was written to memory, then the timing parameter *tRCD* was lowered to 0 clock cycles. Sequential attempts were made to read the data pattern with the results of these attempted read operations forming the PUF response. No error correction or filtering was applied. Data was generated for test patterns 0x00 (all '0'), 0xFF (all '1') and 0x55 (checkerboard pattern). For use in the proposed scheme each DRAM PUF response was transposed to match the required size of our H-matrices. Error correction was not necessary as only a single execution of the obfuscation scheme was required for each given DRAM-PUF measurement, meaning we can assume the DRAM-PUF measurement used is the final 'golden response'.

4.3. Equipment Used

Finally, we performed each of our ML attacks using Python 3.8.12, PyPuf and ScikitLearn on an Intel i9-11980Hk CPU @ 2.60GHz and DDR4 3200MHz 64GB memory. Also, in order to observe the total hardware overhead, we implemented our proposed scheme on a Xilinx Zynq-7000 FPGA device.

4.4. DRAM-PUF Characterisation

As discussed in Section 2.3, it is imperative for the security of the OWF that the supplied H-matrices are cryptographically significant, such that they exhibit ideal uniformity and hamming distance properties (per bit) on the final output. The strong security properties of DRAM based PUFs have been demonstrated experimentally in previous works [23], [32]. In order to verify that the properties seen in previous works held true for the data generated for use in this work, we performed characterisation tests across each unique DRAM-PUF response to determine the suitability for use as an H-matrix. The challenge for the DRAM-PUF can be configured by two means: memory location and input pattern, each combination of which (ideally) produces a unique response output pattern. Further, the OWF output may be configured through incrementing or decrementing the OWF w value, therefore we include this parameter to enable an increased challenge space, which enables unique OWF configurations totalling: *unique memory locations * input patterns (3) * w*. We refer to a combination of H-matrix and w value from now on as the C^{OWF}. We configured the OWF with each unique C^{OWF} and tested the outputs over 10,000, 20,000 and 30,000 (separately) unique samples for uniformity and hamming distance. Practically, this process would occur during the enrollment phase of the scheme, where specific combinations of memory location, input pattern and OWF w value are tested for knowledge in advance of which unique challenged should be used during authentication. We found that the DRAM-Latency PUF exhibits a very strong ability to generate cryptographically significant H-matrices. Across our tests, we found that uniformity was ideal (50 +/- 0.5) in more than 80% of cases, with 81.3% of available unique challenge space (memory locations * w) demonstrating ideal uniformity for challenge pattern 0x00, 88.4% for pattern 0xFF and 100% for pattern 0x55. Furthermore, our results demonstrated that ideal hamming distance is adhered to extremely well, with all C^{OWF}s consisting of an ideal hamming distance of 32 +- 0.5 (for 64 bits). The observed trend of solid pattern (0x00, 0xFF) inputs resulting in reduced uniformity in some memory regions while mixed pattern inputs produce near ideal uniformity is consistent with the results reported in [23] [32].

5. Results and Analysis

In this section, we present the obtained results and discuss our observations.

5.1. Resilience to ML-MA

Here, we first report the findings of classic PUF ML-MA methods on the Strong-PUFs both with and without the proposed scheme. After, we report the findings of our custom ML attack on the scheme. In order to test a variety of PUF types, we performed all tests on both 16-stage and 32-stage variants of an APUF, XOR-APUF and FF-APUF (described in Section 1).

5.1.1. Classical ML-MA. In order to benchmark our proposed scheme against the classic ML-MA methods on PUFs, we first performed both the LR attack proposed in [34] and the MLP attack proposed in [1] on both the PUFs with and without the proposed scheme. For fair comparison, we conducted two experiments. In this first experiment, we perform both the LR and MLP attacks on the Strong-PUFs without the obfuscation scheme, such that we assume an adversary has access to the Strong-PUF challenges and responses, C_{Origin} and R_{Origin}, and aims to predict direct new R_{Origin} responses from the same PUF. In the second experiment, we consider the threat model described in 2.5, where an attacker gains access to the initial challenges and final responses of the obfuscation scheme, C_{Origin} and R_{Final}, and aims to predict new final response bits. We trained each model for the 16-stage APUF on 5000 CRPs and 10,000 CRPs

TABLE 2: ML-MA using Classic Methods

(a) Experiment 1: ML-MA without proposed obfuscation scheme (C_{Origin} -> R_{Origin})

PUF Type	Num Stage	Attack	Training/Testing CRPs	Accuracy
APUF	16	LR	5000	0.944
		MLP	5000	0.988
	32	LR	10000	0.977
		MLP	10000	0.987
XOR-APUF	16	LR	50000	0.974
		MLP	50000	0.989
	32	LR	100000	0.944
		MLP	100000	0.984
FF-APUF	16	LR	50000	0.603
		MLP	50000	0.97
	32	LR	100000	0.645
		MLP	100000	0.981

(b) Experiment 2: ML-MA with proposed obfuscation scheme (C_{Origin} -> R_{Final})

PUF Type	Num Stage	Attack	Training/Testing CRPs	Accuracy
APUF	16	LR	720000	0.53
		MLP	720000	0.531
	32	LR	360000	0.496
		MLP	360000	0.477
XOR-APUF	16	LR	720000	0.525
		MLP	720000	0.529
	32	LR	360000	0.509
		MLP	360000	0.517
FF-APUF	16	LR	720000	0.484
		MLP	720000	0.521
	32	LR	360000	0.498
		MLP	360000	0.478

for the 32-stage APUF. Due to the increased complexity of the XOR-APUF and FF-APUF, for consistency we trained each model on 10x the number of CRPs as the APUF variants, though it is possible to successfully model these PUFs with fewer CRPs as demonstrated in [34] and [1], especially considering they have few stages. Finally, the training CRPs were split into sets of 99% for training and 1% for validation. Table 2(a) provides the details and prediction accuracies of the classical ML-MA attacks. Our benchmark showed expected results, where prediction accuracies for each PUF remained above 94% for each attack, with the exception of the FF-APUF for the LR attack with around 60% accuracy due to the enhanced resilience against linear classifiers. When faced with the MLP attack, the prediction accuracy matched that of the other PUFs at above 97%. However, when the obfuscation scheme is implemented, the modelling results show a significantly reduced performance. Each PUF was attacked using the maximum available CRPs, with 720000 for the 16-stage PUFs and 360000 for the 32-stage PUFs. As larger challenges are required per final response bit, more possible CRPs are available for an attacker to train the model. Table 2(b) shows the results of the classical ML attacks when integrating the proposed scheme. Each attack performed for each PUF showed an average prediction average of 50%. Even when using the most vulnerable 16-bit APUF, the maximum prediction accuracy was extremely low at 53.1%, which is almost the equivalent of a random coin flip. This result demonstrates the potent ability for the scheme to obscure any linearity between original challenge and final response, such that an attacker is provided no advantage when attempting an ML attack.

5.1.2. Custom ML-MA. As the proposed scheme deviates from the simple additive linear delay model of the Strong-PUFs on their own and the number of final output bits available to an attacker, it is also necessary to test a wider variety of classifiers against our proposed scheme. We first benchmark the performance of a set of supervised classification algorithms consisting of Decision Tree, Random Forest, Extra Trees, Logistic Regression, Quadratic Discriminant Analysis, Naive Bayes, Ridge, Light Gradient Boosting Machine, and Linear Discriminant Analysis, the results of which are depicted in Table 3a. We optimise the performance of each ML classifier by performing a random grid search using a predefined internal grid. Each model is evaluated using raw accuracy, AUC, recall, precision, F1, Kappa and MCC [5], [16]. For brevity, the details of each metric have been provided in Appendix 1 for the interested reader.

We also benchmark the performance of our proposed scheme against a set of unsupervised algorithms, namely Deep Q-Network, HDBSCAN, DBSCAN, BIRCH, K-Means++, K-Means, K-Medoids, Gaussian. Unsupervised classifiers, where obscure connections in an unlabeled dataset are discovered by grouping data into clusters or by association. Most importantly, we evaluate our proposed scheme against a reinforcement learning (RL) based attacker using Deep Q-Networks (DQN). DQN is an approximation-based RL where an agent interacts with the environment by sensing its state and learns to take action to maximise long-term reward. As the agent takes action, it needs to maintain a balance between exploration and exploitation by performing a variety of actions using trial and error in an uncertain environment to favour the actions that yield the maximum reward in the future. This type of exploratory attack is well suited for learning highly non-linear relationships/correlations within a complex feature space, such as is required for modelling PUF obfuscation schemes, similar to the CMA-ES attack performed in [3]. The results of the unsupervised classifiers is shown in Table 3b

In order to ensure the generalisation and verify results, we perform 10-fold-cross validations on each experiment experiment. The challenges (training features) consisted of 1024 and 2048 features (for each bit) for the 16-stage and 32-stage PUF respectively. As the scheme outputs 4 bits for the 16-stage Strong PUFs and 2 bits for the 32-stage Strong PUFs, the classification labels were 4 and 2 bits respectively (15 and 4 labels).

Our results showed each classifier faced extreme difficulty in detecting a relationship between the challenge and response data for both the supervised and unsupervised classifiers, with the accuracy of each classifier for each PUF type not exceeding 6.3% for the 16-bit PUFs and 20% for the 32-bit PUFs. Overall, the unsupervised classifiers performed slightly better overall than the supervised methods, which is intuitive given the suitability of many unsupervised classification methods to non-linear predictive tasks such as required for PUF obfuscation modelling.

TABLE 3: Custom ML-MA Results

Model	PUF	Num. Stages	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Logistic Regression	APUF	16	0.0630	0.5021	0.0629	0.0630	0.0629	0.0005	0.0005
		32	0.2485	0.4979	0.2483	0.2483	0.2482	-0.0023	-0.0023
	XOR-APUF	16	0.0640	0.4991	0.0638	0.0638	0.0638	0.0014	0.0014
		32	0.2485	0.4979	0.2483	0.2483	0.2482	-0.0023	-0.0023
	FF-APUF	16	0.0621	0.5002	0.0621	0.0621	0.0620	-0.0005	-0.0005
		32	0.2524	0.5014	0.2524	0.2524	0.2524	0.0031	0.0031
Linear Discriminant Analysis	APUF	16	0.0629	0.5021	0.0628	0.0629	0.0628	0.0003	0.0003
		32	0.2484	0.4979	0.2481	0.2481	0.2480	-0.0025	-0.0025
	XOR-APUF	16	0.0638	0.4992	0.0636	0.0636	0.0635	0.0012	0.0012
		32	0.2484	0.4979	0.2481	0.2481	0.2480	-0.0025	-0.0025
	FF-APUF	16	0.0556	0.4505	0.0556	0.0555	0.0555	-0.0007	-0.0007
		32	0.2522	0.5014	0.2522	0.2522	0.2522	0.0029	0.0029
Quadratic Discriminant Analysis	APUF	16	0.0627	0.5000	0.0626	0.0624	0.0513	0.0000	0.0001
		32	0.2480	0.4970	0.2477	0.2478	0.2476	-0.0030	-0.0030
	XOR-APUF	16	0.0622	0.4987	0.0617	0.0617	0.0612	-0.0008	-0.0008
		32	0.2480	0.4970	0.2477	0.2478	0.2476	-0.0030	-0.0030
	FF-APUF	16	0.0620	0.4997	0.0620	0.0615	0.0452	-0.0006	-0.0005
		32	0.2506	0.5005	0.2506	0.2506	0.2506	0.0008	0.0008
Ridge Classifier	APUF	16	0.0627	0.0000	0.0627	0.0627	0.0625	0.0002	0.0002
		32	0.2484	0.0000	0.2481	0.2481	0.2480	-0.0025	-0.0025
	XOR-APUF	16	0.0637	0.0000	0.0635	0.0635	0.0633	0.0011	0.0011
		32	0.2484	0.0000	0.2481	0.2481	0.2480	-0.0025	-0.0025
	FF-APUF	16	0.0619	0.0000	0.0618	0.0618	0.0617	-0.0007	-0.0007
		32	0.2522	0.0000	0.2522	0.2522	0.2522	0.0029	0.0029
Naïve Bayes	APUF	16	0.0629	0.5023	0.0628	0.0629	0.0628	0.0004	0.0004
		32	0.2494	0.4982	0.2492	0.2492	0.2491	-0.0011	-0.0011
	XOR-APUF	16	0.0634	0.4994	0.0632	0.0633	0.0632	0.0008	0.0008
		32	0.2494	0.4982	0.2492	0.2492	0.2491	-0.0011	-0.0011
	FF-APUF	16	0.0619	0.5004	0.0619	0.0619	0.0618	-0.0006	-0.0006
		32	0.2527	0.5013	0.2527	0.2528	0.2527	0.0037	0.0037
Decision Tree	APUF	16	0.0626	0.5001	0.0626	0.0626	0.0626	0.0001	0.0001
		32	0.2482	0.4988	0.2482	0.2482	0.2482	-0.0024	-0.0024
	XOR-APUF	16	0.0632	0.5003	0.0631	0.0631	0.0631	0.0007	0.0007
		32	0.2482	0.4988	0.2482	0.2482	0.2482	-0.0024	-0.0024
	FF-APUF	16	0.0639	0.5007	0.0638	0.0639	0.0639	0.0014	0.0014
		32	0.2495	0.4997	0.2495	0.2495	0.2495	-0.0007	-0.0007
Random Forrest	APUF	16	0.0623	0.4997	0.0622	0.0622	0.0617	-0.0003	-0.0003
		32	0.2509	0.4983	0.2509	0.2510	0.2509	0.0012	0.0012
	XOR-APUF	16	0.0626	0.4998	0.0627	0.0626	0.0624	0.0002	0.0002
		32	0.2509	0.4983	0.2509	0.2510	0.2509	0.0012	0.0012
	FF-APUF	16	0.0630	0.5013	0.0630	0.0629	0.0624	0.0005	0.0005
		32	0.2500	0.5002	0.2500	0.2499	0.2495	-0.0001	-0.0001
Extra Trees Classifier	APUF	16	0.0615	0.4988	0.0614	0.0614	0.0609	-0.0011	-0.0011
		32	0.2499	0.4990	0.2499	0.2499	0.2499	-0.0002	-0.0002
	XOR-APUF	16	0.0624	0.4998	0.0625	0.0625	0.0623	0.0000	0.0000
		32	0.2499	0.4990	0.2499	0.2499	0.2499	-0.0002	-0.0002
	FF-APUF	16	0.0625	0.4988	0.0624	0.0623	0.0617	-0.0001	-0.0001
		32	0.2510	0.5021	0.2509	0.2510	0.2505	0.0012	0.0012

(a) Custom supervised ML-MA results

Model	PUF	Num. Stages	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
DQN	APUF	16	0.0637	0.0000	0.0636	0.0635	0.0634	0.0012	0.0012
		32	0.2523	0.5000	0.2522	0.2523	0.2523	0.0030	0.0030
	XOR-APUF	16	0.0639	0.0000	0.0635	0.0634	0.0633	0.0012	0.0012
		32	0.2532	0.5021	0.2532	0.2533	0.2532	0.0042	0.0042
	FF-APUF	16	0.0636	0.5012	0.0635	0.0634	0.0632	0.0011	0.0011
		32	0.2540	0.5026	0.2539	0.2539	0.2538	0.0052	0.0052
HDBSCAN	APUF	16	0.0635	0.4995	0.0635	0.0634	0.0634	0.0010	0.0010
		32	0.2517	0.5002	0.2511	0.2511	0.2505	0.0015	0.0015
	XOR-APUF	16	0.0639	0.5012	0.0637	0.0637	0.0637	0.0014	0.0014
		32	0.2511	0.5006	0.2509	0.2510	0.2509	0.0012	0.0012
	FF-APUF	16	0.0628	0.5006	0.0627	0.0629	0.0622	0.0003	0.0003
		32	0.2531	0.0000	0.2531	0.2531	0.2531	0.0041	0.0041
DBSCAN	APUF	16	0.0632	0.4996	0.0632	0.0631	0.0631	0.0007	0.0007
		32	0.2510	0.5001	0.2509	0.2510	0.2509	0.0012	0.0012
	XOR-APUF	16	0.0637	0.4993	0.0630	0.0629	0.0616	0.0005	0.0005
		32	0.2494	0.4982	0.2486	0.2484	0.2476	-0.0019	-0.0019
	FF-APUF	16	0.0621	0.5003	0.0621	0.0621	0.0620	-0.0004	-0.0004
		32	0.2531	0.5026	0.2530	0.2531	0.2530	0.0041	0.0041
BIRCH	APUF	16	0.0632	0.4993	0.0632	0.0631	0.0631	0.0007	0.0007
		32	0.2491	0.4990	0.2488	0.2489	0.2488	-0.0016	-0.0016
	XOR-APUF	16	0.0636	0.5011	0.0634	0.0633	0.0633	0.0010	0.0010
		32	0.2490	0.5012	0.2488	0.2489	0.2489	-0.0016	-0.0016
	FF-APUF	16	0.0620	0.0000	0.0619	0.0619	0.0618	-0.0006	-0.0006
		32	0.2530	0.5025	0.2530	0.2530	0.2529	0.0040	0.0040
K-Means++	APUF	16	0.0631	0.5008	0.0630	0.0633	0.0627	0.0006	0.0006
		32	0.2491	0.0000	0.2488	0.2489	0.2488	-0.0016	-0.0016
	XOR-APUF	16	0.0636	0.5013	0.0634	0.0634	0.0634	0.0010	0.0010
		32	0.2483	0.4985	0.2479	0.2480	0.2478	-0.0029	-0.0029
	FF-APUF	16	0.0618	0.4996	0.0618	0.0619	0.0618	-0.0008	-0.0008
		32	0.2524	0.5023	0.2524	0.2524	0.2524	0.0032	0.0032
K-Means	APUF	16	0.0625	0.5015	0.0622	0.0622	0.0617	-0.0003	-0.0003
		32	0.2490	0.4987	0.2487	0.2488	0.2487	-0.0017	-0.0017
	XOR-APUF	16	0.0635	0.5005	0.0635	0.0635	0.0635	0.0010	0.0010
		32	0.2474	0.0000	0.2470	0.2471	0.2469	-0.0040	-0.0040
	FF-APUF	16	0.0617	0.5000	0.0617	0.0617	0.0616	-0.0009	-0.0009
		32	0.2504	0.5016	0.2503	0.2506	0.2499	0.0004	0.0005
K-Medoids	APUF	16	0.0614	0.4986	0.0613	0.0611	0.0607	-0.0012	-0.0012
		32	0.2486	0.4987	0.2484	0.2485	0.2484	-0.0022	-0.0022
	XOR-APUF	16	0.0630	0.5008	0.0630	0.0630	0.0627	0.0005	0.0005
		32	0.2474	0.4978	0.2470	0.2471	0.2470	-0.0040	-0.0040
	FF-APUF	16	0.0617	0.4996	0.0617	0.0617	0.0611	-0.0009	-0.0009
		32	0.2502	0.5009	0.2502	0.2502	0.2497	0.0002	0.0002
Gaussian	APUF	16	0.0612	0.4993	0.0612	0.0612	0.0611	-0.0014	-0.0014
		32	0.2479	0.4985	0.2478	0.2479	0.2478	-0.0029	-0.0029
	XOR-APUF	16	0.0625	0.4997	0.0625	0.0626	0.0623	0.0000	0.0000
		32	0.2473	0.4976	0.2469	0.2470	0.2469	-0.0041	-0.0041
	FF-APUF	16	0.0617	0.5002	0.0617	0.0616	0.0616	-0.0009	-0.0009
		32	0.2488	0.4992	0.2488	0.2488	0.2487	-0.0017	-0.0017

(b) Custom unsupervised ML-MA results

5.2. Security Analysis

In this section, we will briefly discuss various potential threats to the proposed system and the implications of those threats. It should be noted that the claims being made in this paper relate to security against remote ML-MA only, as defined in the threat model in Section 2.5. Resistance to attacks requiring physical access is left for future work. Nonetheless, we will discuss some such attacks in order to provide the reader with a clear picture of where this countermeasure fits into the overall threat landscape.

5.2.1. ML-MA Remote Attacks. As shown in Section 5.1, the countermeasures in the scheme render modeling the system as a whole infeasible if only the overall inputs and outputs are known and not the internal states. These inputs and outputs are the only items of information transmitted over open channels; therefore, this strongly impedes remote modeling attacks.

5.2.2. Replay Attacks. As the challenges and responses are being transmitted openly, and it is explicitly assumed the adversary can listen in on this channel, there is a requirement at the protocol level to invalidate CRPs once they have been used. Otherwise, there is a risk that if a previously used challenge is issued for which the adversary recorded the response, they would be able to provide the correct response despite no knowledge of the PUF. There is a similar need to prevent an adversary from issuing rapid false challenges to gather the full CRP set through rate-limiting challenges at the device side, removing CRPs if a response is received unexpectedly or both.

5.2.3. Cold Boot Attacks. As the scheme uses a memory PUF to generate matrices for the OWF, attacks which aim to capture data from volatile memories such as cold boot attacks have the potential to compromise the matrices. How dangerous is this in practice? Consider a scenario where the adversary knows the n column matrix, H, the OWF padding method, the OWF block number, w, and the output of the OWF, R_{Final}. They want to learn the OWF input (i.e. the raw PUF output). To reverse the OWF with this information requires them to guess which columns are XORed, which will require n^w trials. e.g. using the value of $n = 1024$ in our example scheme and $w = 16$ would require in the order of 2^{160} trials. Thus, H being known to the adversary *does* reduce the complexity of OWF reversal, but not enough to meaningfully compromise the system.

5.2.4. Memory Snooping Attacks. Attacks which allow snooping of system memory or manipulation of the memory controller (e.g. malicious software) could conceivably

leak the matrix set, which is generated from memory and resides temporarily in it. The same analysis as the previous point applies here, where knowing the matrices isn't enough in itself to compromise the system. The only other aspects which ever reside in system memory are the challenges and final responses which we assume the adversary can acquire over the transmission channel anyway.

5.2.5. Timing Side-Channel Attacks. While the OWF itself is time-invariant, this is not necessarily true for the PUF, or for the PUF error correction. In fact, *because* the OWF is fixed time the timing of the whole scheme reveals information about the timing of the PUF+ECC block. This information could conceivably be used to assist model construction and make the scheme easier to compromise, although the exact efficacy of such an attack is uncertain. Ideally, the PUF block and ECC should be made time-invariant, which removes this channel of information leakage. This is especially relevant to this work as timing can be analysed remotely to a degree, unlike other side channels, which require physical access.

5.2.6. EM Side Channel Attacks. If the adversary has physical access to the target device there may be attacks which could be employed using EM side-channel analysis. As with the other side-channel attacks, there are two separate risks. First, direct measurement of the PUF as it operates. Second, the measurement of OWF internal states breaks the security properties of the function. The actual degree of leakage through EM emission and how to mask it if necessary are outside the scope of this work, but it should be noted as a factor for consideration in future work.

5.2.7. Voltage Side Channel Attacks. As with the EM emissions, there may be some information which can be derived about the internal states through monitoring of power consumption during operation. Again, exploration of this, and how to level out power use in the scheme if necessary, is outside scope of this work but may be explored in future works.

5.2.8. Fault Injection Attacks. The points above assume correct operation but there may be additional risks if the adversary can intentionally cause faults in parts of the system. For example, it may be possible to partially bypass the complexity of the attack mentioned in Section 5.2.3 by selectively faulting one column of H at a time while repeatedly issuing the same challenge. Or issuing falsified challenges and selectively faulting the responses such that the protocol does not invalidate the collected CRP. Such attacks unequivocally require prolonged physical access and are outside the scope of this work, but they provide interesting possibilities for future work and are worth noting.

5.3. Hardware Overhead

Strengths and weaknesses based on certain included features tend to map linearly to required hardware overhead, as demonstrated by similar obfuscation schemes. The hardware overhead of our scheme includes the Strong-PUF utilised, PUF error correction, 64-bit buffer and the OWF. Different types and/or sizes of Strong PUFs will incur varying hardware overhead; however, in our scheme, some overhead is mitigated through the use of smaller-length PUFs. Comparable schemes such as [42] employ 64 to 128-bit Arbiter PUFs, whereas our scheme demonstrated protection for even 32-stage PUFs. Due to the suitability of our scheme with APUFs, we benchmark our scheme using the hardware overhead of 16-bit and 32-bit APUFs. As we propose a generic obfuscation scheme, the required ECC will vary as different Strong-PUFs are used for a token generation as different PUF types incur different reliability properties [19] which can have varying effects on the overall hardware overhead of the scheme. PUFs tend to exhibit a higher error rate as the size and complexity of the circuitry grow, for example, the more individual APUFs used to construct an XOR-APUF, the higher the error rate tends to be. The APUF implementation in [18] can achieve a Bit Error Rate (BER) of as low as 10^{-9}, meaning lower cost ECC schemes that rely on high PUF reliability such as provided by Hiller et al. in [20] may be implemented. Therefore, given our proposed benchmark, it is possible to utilise a highly reliable APUF and thus employ far less resource-consuming ECC methods. For a more comprehensive ECC implementation, we separately performed majority voting [7] and the Golay (23,12,7) code [26] for both 16-stage and 32-stage variants of an APUF. We performed a synthesis on the programmable logic of a Xilinx Zynq-7000 FPGA to determine the hardware overhead. Figure 3 shows the chip layout for the synthesised generic PUF obfuscation scheme. The total resource requirements of our scheme are listed in Table 4.

TABLE 4: Comparison of the Hardware and Power Overhead of the Proposed Scheme Against the State-of-the-art

Scheme	LUTs	DFFs	Power (*W*)
Controlled PUF [13]	1830	3020	∼
PUF-FSM [11]	960	1500	∼
Set-based [42]	395	1400	∼
CT-PUF [43]	741	486	0.107
Proposed Scheme			
APUF: † **ECC:** ‡‡	306	298	0.177
APUF: † **ECC:** ††	291	330	0.178
APUF: ‡ **ECC:** ‡‡	341	312	0.180
APUF: ‡ **ECC:** ††	327	345	0.179

† 16-Bit ‡ 32-Bit †† Golay Code ‡‡ Majority Voting
∼ Data not available

5.4. Power Consumption

As the memory PUF operates on existing components, namely the ARM Cortex A9 processor cores of the Zynq and an external DDR3 DRAM chip, power consumption for it is a function of how much additional load the required instructions add to the Processor System (PS). Each matrix generation requires a number of memory accesses that scales proportionally with the number of rounds of majority vote ECC. For matrices of the size used in our experiments this requires 2048 writes and $2048m$ reads to the DDR memory, where m is the number of rounds. This uses negligible power even for fairly large m (e.g. at $m = 100$ memory access load is increased by 0.034%,

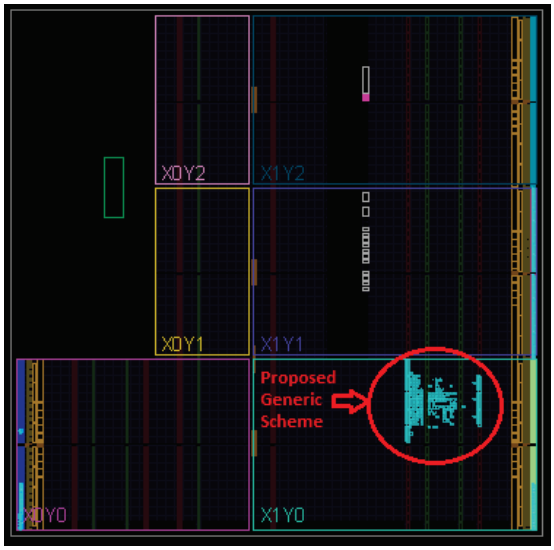

Figure 3: Layout of the Proposed Generic PUF Obfuscation Scheme (16-bit APUF and Majority Vote ECC) on a Xilinx Zynq-7000 FPGA Device

with power consumption of $< 1mW$). Error correction also requires $(258n)\times m$ processor instructions, where n is the bit length of the matrix. For the example matrices and 10 round ECC this equates to 26% of maximum processor load for one core, which consumes $0.065W$. For the FPGA implementation, we utilised Xilinx power estimator to determine the power consumption of the part of the scheme which operates on the programmable logic. We found the total power consumption to be very low, using only $0.112W$ at minimum (16-bit APUF and Majority Vote ECC) and $0.115W$ at maximum (32-bit APUF and Majority Vote ECC). From this, we can simply calculate the total power consumption of one full cycle of the scheme by adding the DRAM-PUF power consumption ($0.065W$) to the FPGA power consumption, which gives a total (maximum) of $0.180W$. Table 4 shows the combined total power cost for different configurations of the overall proposed scheme (FPGA logic and DRAM-PUF).

5.5. Implications of Varying Hardware

It should be noted that in addition to the costs listed there is a variable degree of RAM usage. The footprint of this depends on the memory configuration of the system. When using the Latency DRAM-PUF as in our experiments, an amount of memory is used during matrix generation at least ten times the size of the desired matrix. This is due to the fact that when using the Latency DRAM-PUF, the actual stored data is unaffected by the PUF process, rather, when the memory is read, the controller misreads what is actually stored and that error pattern is the PUF response. Therefore, when using majority vote error correction you need an area the size of H to perform the PUF operation on, another equally sized area to store the temporary result, and at least one byte per bit of that for per-bit majority vote counting. This is fine so long as there is sufficient DRAM free to allow a region of that size to be available whenever the PUF needs to be queried. If, however, there is no DRAM available or this overhead is not practical, BRAMs on the FPGA logic can be utilised to store matrices instead, at the cost of increasing the overall FPGA hardware footprint of the scheme. Other Memory-PUFs that cause changes directly in the stored data, such as Start-up SRAM-PUF and Retention DRAM-PUF, can also be used and would have a smaller footprint as the matrix is generated directly in the region of memory being used as the PUF, though there may be some overhead required for error correction. However, these PUFs are not as easy to use in-runtime due to their query processes being destructive to data held in the same memory. Given these factors, a trade-off is available to system designers based on which features are most desirable for the given application:

Latency DRAM-PUF:
- \+ Very fast measurement speed
- \+ Large space available for many supported unique responses
- \+ Non-destructive to memory contents
- – Additional memory space required to store response

Retention DRAM-PUF:
- \+ No additional memory required to store response
- \+ Large space available for many supported unique responses
- – Very slow measurement speed
- – Destructive to memory contents

Start-up SRAM-PUF:
- \+ Fast measurement speed
- \+ No additional memory space required to store response
- – Lower density therefore fewer unique responses supported
- – Requires power cycle to query
- – Destructive to memory contents

6. Conclusion

Physical Unclonable Functions (PUFs) offer a promising solution for the lightweight authentication of IoT devices as they provide unique fingerprints for the underlying devices through their challenge-response pairs. However, PUFs have been shown to be vulnerable to Machine-Learning Modelling-Attacks (ML-MA). In this article, we propose a novel obfuscation scheme for preventing ML-MA on Strong PUFs by exploiting readily-available device resources in the form of a Memory-PUF and a One-Way Function to obfuscate PUF challenges and responses. We demonstrate our scheme has a significant effect on reducing the accuracy of ML-MA to almost the same probability as a random coin flip when tested against various established attacks from the literature and against our own additionally tested classifiers. Our experimental results also show our scheme has the potential to be very cost-effective with regards to hardware overhead on FPGA-enabled devices. While this work focused on security against ML-MA, side-channel/fault injection attacks are also potential attack vectors for PUF schemes where attackers can gain physical access to devices, which may form the basis of future work.

Data Availability

For reproducibility of our work and engagement with the wider research community, all source code and data for our experiments have been made publicly available at the following link: https://drive.google.com/drive/folders/1ZWgkjkicVNUYB3cRUnfV0mOxvZhOyLHz

Acknowledgements

The Royal Society Research Grant supported the work of Prosanta Gope under grant RGS\R1\221183. The research was partially supported by the Engineering and Physical Sciences Research Council SIPP project (grant number EP/S030867/1). Meltem Kurt Pehlivanoğlu is partially supported by The Scientific and Technological Research Council of Turkey (TUBITAK) under the 2219 Postdoctoral Research Program Grant. The authors would like to thank Dr Chenghua Lin for supporting this work through technical expertise, advice and feedback and Mr Ping-Chen Lin (BSc) for helping create code for performing PUF simulation experiments.

References

[1] Mohammed Saeed Alkatheiri and Yu Zhuang. Towards fast and accurate machine learning attacks of feed-forward arbiter pufs. In *2017 IEEE Conference on Dependable and Secure Computing*, pages 181–187, 2017.

[2] Daniel Augot, Matthieu Finiasz, and Nicolas Sendrier. Improved fast syndrome based cryptographic hash functions. In *in Proceedings of ECRYPT Hash Workshop 2007 (2007). URL: http://www-roc.inria.fr/secret/Matthieu.Finiasz*.

[3] Georg Becker. On the pitfalls of using arbiter pufs as building blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34:1–1, 08 2015.

[4] E. Berlekamp, R. McEliece, and H. van Tilborg. On the inherent intractability of certain coding problems (corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, 1978.

[5] Kathrin Blagec, Georg Dorffner, Milad Moradi, and Matthias Samwald. A critical analysis of metrics used for measuring progress in artificial intelligence, 2020.

[6] Yijun Cui, Chongyan Gu, Qingqing Ma, Yue Fang, Chenghua Wang, Máire O'Neill, and Weiqiang Liu. Lightweight modeling attack-resistant multiplexer-based multi-puf (mmpuf) design on fpga. *Electronics*, 9(5):815, 2020.

[7] Abhishek Das and Nur A. Touba. A single error correcting code with one-step group partitioned decoding based on shared majority-vote. *Electronics*, 9(5), 2020.

[8] Jeroen Delvaux. Machine-learning attacks on polypufs, ob-pufs, rpufs, lhs-pufs, and puf–fsms. *IEEE Transactions on Information Forensics and Security*, 14(8):2043–2058, 2019.

[9] Elena Dubrova, Oscar Näslund, Bernhard Degen, Anders Gawell, and Yang Yu. Crc-puf: A machine learning attack resistant lightweight puf construction. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroSPW)*, pages 264–271, 2019.

[10] Matthieu Finiasz. Syndrome based collision resistant hashing. In Johannes Buchmann and Jintai Ding, editors, *Post-Quantum Cryptography*, pages 137–147, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[11] Yansong Gao, Hua Ma, Said F. Al-Sarawi, Derek Abbott, and Damith C. Ranasinghe. Puf-fsm: A controlled strong puf. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):1104–1108, 2018.

[12] Blaise Gassend, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Silicon Physical Random Functions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 148–160, New York, NY, USA, 2002. Association for Computing Machinery.

[13] Blaise Gassend, Marten Van Dijk, Dwaine Clarke, Emina Torlak, Srinivas Devadas, and Pim Tuyls. Controlled physical random functions and applications. *ACM Trans. Inf. Syst. Secur.*, 10(4), jan 2008.

[14] Blaise Gassend, Daihyun Lim, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098, 2004.

[15] Prosanta Gope, Owen Millwood, and Biplab Sikdar. A Scalable Protocol Level Approach to Prevent Machine Learning Attacks on Physically Unclonable Function Based Authentication Mechanisms for Internet of Medical Things. *IEEE Transactions on Industrial Informatics*, 18(3):1971–1980, 2022.

[16] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview, 2020.

[17] Zhangqing He, Wanbo Chen, Lingchao Zhang, Gaojun Chi, Qi Gao, and Lein Harn. A highly reliable arbiter puf with improved uniqueness in fpga implementation using bit-self-test. *IEEE Access*, 8:181751–181762, 2020.

[18] Zhangqing He, Wanbo Chen, Lingchao Zhang, Gaojun Chi, Qi Gao, and Lein Harn. A highly reliable arbiter puf with improved uniqueness in fpga implementation using bit-self-test. *IEEE access*, 8:181751–181762, 2020.

[19] Matthias Hiller, Ludwig Kürzinger, and Georg Sigl. Review of error correction for pufs and evaluation on state-of-the-art fpgas. *Journal of cryptographic engineering*, 10(3):229–247, 2020.

[20] Matthias Hiller, Meng-Day (Mandel) Yu, and Michael Pehl. Systematic low leakage coding for physical unclonable functions. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, page 155–166, New York, NY, USA, 2015. Association for Computing Machinery.

[21] Daniel E. Holcomb, Wayne P. Burleson, and Kevin Fu. Power-up sram state as an identifying fingerprint and source of true random numbers. *IEEE Transactions on Computers*, 58(9):1198–1210, 2009.

[22] C. Keller, F. Gürkaynak, H. Kaeslin, and N. Felber. Dynamic memory-based physically unclonable function for the generation of unique identifiers and true random numbers. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2740–2743, 2014.

[23] Jeremie S Kim, Minesh Patel, Hasan Hassan, and Onur Mutlu. The dram latency puf: Quickly evaluating physical unclonable functions by exploiting the latency-reliability tradeoff in modern commodity dram devices. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, volume 2018-, pages 194–207. IEEE, 2018.

[24] Vlastimil Klima and Tomas Rosa. Side channel attacks on cbc encrypted messages in the pkcs7 format. Cryptology ePrint Archive, Paper 2003/098, 2003. https://eprint.iacr.org/2003/098.

[25] Raghavan Kumar and Wayne Burleson. On design of a highly secure puf based on non-linear current mirrors. In *2014 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 38–43, 2014.

[26] Hung-Peng Lee, Shao-I Chu, and Hsin-Chiu Chang. Efficient decoding of the (23, 12, 7) golay code up to five errors. *Information Sciences*, 253:170–178, 2013.

[27] J.W. Lee, Daihyun Lim, B. Gassend, G.E. Suh, M. van Dijk, and S. Devadas. A technique to build a secret key in integrated circuits for identification and authentication applications. In *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, pages 176–179, 2004.

[28] Qingqing Ma, Chongyan Gu, Neil Hanley, Chenghua Wang, Weiqiang Liu, and Maire O'Neill. A machine learning attack resistant multi-puf design on fpga. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 97–104. IEEE, 2018.

[29] Mehrdad Majzoobi, Farinaz Koushanfar, and Miodrag Potkonjak. Lightweight secure pufs. In *2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673, 2008.

[30] Owen Millwood, Jack Miskelly, Bohao Yang, Prosanta Gope, Elif Bilge Kavun, and Chenghua Lin. Puf-phenotype: A robust and noise-resilient approach to aid group-based authentication with dram-pufs using machine learning. *IEEE Transactions on Information Forensics and Security*, pages 1–1, 2023.

[31] Jack Miskelly, Chongyan Gu, Qingqing Ma, Yijun Cui, Weiqiang Liu, and Máire O'Neill. Modelling attack analysis of configurable ring oscillator (cro) puf designs. In *2018 IEEE 23rd international conference on digital signal processing (DSP)*, pages 1–5. IEEE, 2018.

[32] Jack Miskelly and Maire O'Neill. Fast dram pufs on commodity devices. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 39(11):3566–3576, 2020.

[33] Phuong Ha Nguyen, Durga Prasad Sahoo, Chenglu Jin, Kaleel Mahmood, Ulrich Rührmair, and Marten van Dijk. The interpose puf: Secure puf design against state-of-the-art machine learning attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):243–290, Aug. 2019.

[34] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber. Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, page 237–249, New York, NY, USA, 2010. Association for Computing Machinery.

[35] Ulrich Rührmair, Jan Sölter, Frank Sehnke, Xiaolin Xu, Ahmed Mahmoud, Vera Stoyanova, Gideon Dror, Jürgen Schmidhuber, Wayne Burleson, and Srinivas Devadas. Puf modeling attacks on simulated and silicon data. *IEEE Transactions on Information Forensics and Security*, 8(11):1876–1891, 2013.

[36] G. Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14, 2007.

[37] Nils Wisiol, Georg T. Becker, Marian Margraf, Tudor A. A. Soroceanu, Johannes Tobisch, and Benjamin Zengin. Breaking the lightweight secure puf: Understanding the relation of input transformations and machine learning resistance. Cryptology ePrint Archive, Report 2019/799, 2019. https://eprint.iacr.org/2019/799.

[38] Nils Wisiol, Christoph Gräbnitz, Christopher Mühl, Benjamin Zengin, Tudor Soroceanu, Niklas Pirnay, Khalid T. Mursi, and Adomas Baliuka. pypuf: Cryptanalysis of Physically Unclonable Functions, 2021.

[39] Nils Wisiol, Christopher Mühl, Niklas Pirnay, Phuong Ha Nguyen, Marian Margraf, Jean-Pierre Seifert, Marten van Dijk, and Ulrich Rührmair. Splitting the interpose puf: A novel modeling attack strategy. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):97–120, Jun. 2020.

[40] Jing Ye, Yu Hu, and Xiaowei Li. Rpuf: Physical unclonable function with randomized challenge to resist modeling attack. In *2016 IEEE Asian Hardware-Oriented Security and Trust (AsianHOST)*, pages 1–6, 2016.

[41] Meng-Day Yu, Matthias Hiller, Jeroen Delvaux, Richard Sowell, Srinivas Devadas, and Ingrid Verbauwhede. A Lockdown Technique to Prevent Machine Learning on PUFs for Lightweight Authentication. *IEEE Transactions on Multi-Scale Computing Systems*, 2(3):146–159, 2016.

[42] Jiliang Zhang and Chaoqun Shen. Set-based obfuscation for strong pufs against machine learning attacks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(1):288–300, 2021.

[43] Jiliang Zhang, Chaoqun Shen, Zhiyang Guo, Qiang Wu, and Wanli Chang. Ct puf: Configurable tristate puf against machine learning attacks for iot security. *IEEE Internet of Things Journal*, 9(16):14452–14462, 2022.

Appendix

1. Evaluation Metrics

Accuracy, shown in Equation 4, is the raw percentage of all labels which are correctly classified, without further inference (*TP*: True Positive, *TN*: True Negative, *FP*: False Positive, *FN*: False Negative). Precision, shown in Equation 5, denotes the amount of all positive classifications which were correctly predicted. Precision is focused on positive classifications, whilst accuracy considers both positive and negative classifications. The F1-measure, shown in Equation 6, combines both precision and recall into a single measure which captures both properties and provides an overall classifier performance.

$$\textit{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

$$\textit{Precision} = \frac{TP}{TP + FP} \tag{5}$$

$$\textit{F1-measure } (\beta = 1) = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{6}$$

The Kappa metric is a measure for evaluating the prediction performance of incremental classifiers. The kappa is computed as Equation 7, where Θ is the accuracy rate of an intelligent classifier and Θ_r is the accuracy rate of a random classifier, which randomly permutes the predictions of the intelligent classifier. The kappa-statistic takes values between 0 and 1, where 0 indicates that the achieved accuracy is random.

$$\textit{Kappa} = \frac{\Theta - \Theta_r}{1 - \Theta_r} \tag{7}$$

Recall or True Positive Rate (*TPR*) is another widely used metric, shown in Equation 8, denoting the percentage of data samples that the model correctly identifies as belonging to a the positive class.

$$\textit{Recall} = \textit{TPR} = \frac{TP}{TP + FN} \tag{8}$$

The Matthews correlation coefficient (MCC), shown in Equation 9 is a reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TP + FP)(TN + FN)}} \tag{9}$$

The AUC metric provides a way to quantify the overall performance of classification models, taking into account the trade-off between sensitivity (true positive rate) and specificity (true negative rate) at different classification thresholds. The AUC metric ranges from 0 to 1, with higher values indicating better model performance. A perfect classifier would have an AUC score of 1, while a completely random classifier would have an AUC score of 0.5